CS522 - Programming Language Semantics

Axiomatic Semantics — Hoare Logic

Grigore Roşu

Department of Computer Science
University of Illinois at Urbana-Champaign

Using equational, rewriting, SOS or denotational semantics we have been able to "execute" programs within their semantics. Moreover, since in some cases we were also able to prove equivalence of semantics, we have a high confidence that each of the semantics is a correct definition of the language under consideration.

However, so far we have investigated no method that allows us to prove programs correct. We have not even formally defined what it means for a program to be correct. We next informally discuss a standard technique to define and prove correctness of programs, called *Hoare logic*, which is quite successful in the context of non-concurrent programs that do not use the heap.

Suppose that one wants to show that the program

```
s = 0 ; n = 1 ;
while (n <= p) (
  s = s + n ;
  n = n + 1
)
```

indeed calculates the sum of the first p natural numbers, that is, that $s = p * (p + 1)/2$ at the end of its execution. A first observation is that this program calculates the correct sum only if run in a state in which p denotes a positive number (technically speaking, it is also correct for $p = -1$, but we ignore this unnecessary generality). Thus, in order to state the correctness of this program, we would like to say the following:

> If executed in a state in which $p \geq 0$ then it is the case that $s = p * (p + 1)/2$ in the state obtained at the end of the execution of this program.

As will be discussed later, the statement above can be formally written as

```
{p >= 0}
  s = 0 ; n = 1 ;
  while (n <= p) (
    s = s + n ;
    n = n + 1
  )
{s == p * (p + 1) / 2}
```

where `{p >= 0}` and `{s == p * (p + 1) / 2}` are *assertions*, the first called *pre-condition* and the second called *post-condition*. In this lecture, to avoid syntactic conflicts, we will enclose blocks by normal parentheses rather than curly brackets.

State Assertions

The assertions above are nothing but boolean expressions, using the syntax that we have already defined for our simple imperative language. However, in general one may need to state more complex state properties, such as, for example, $(\exists k)x = 2^k$, saying that x is a power of 2.

Our goal in this lecture is to understand the major underlying concepts of Hoare logic and the basic intuitions for what is called *theorem proving* in software analysis/verification, so we will *not* formalize rigorously the syntax and semantics of assertions.

Intuitively, besides the usual boolean expression operators, we also allow *quantifiers* with the syntax $(\forall\langle IntVar\rangle)\ \langle Assertion\rangle$ and $(\exists\langle IntVar\rangle)\ \langle Assertion\rangle$. There is an intuitive notion of *validity* of assertions, saying that an assertion is valid if and only if it holds in any possible state. For example, $(\forall i)(x > i \vee x = i \vee x < i)$ is valid.

The famous *Gödel's incompleteness theorem* tells us that there is no way to find an algorithm able to prove any valid assertion, which tells us that there is no way to prove *any* property one may want to prove about a program.

Indeed, if one had an algorithm to prove any property about a program, then, given any assertion `A`, one may use that algorithm to prove that `{} skip {A}`, which is equivalent to proving `A`. Thus, an algorithm able to prove any property about programs would immediately translate into an algorithm to prove any valid assertion, which contradicts Gödel's incompleteness theorem.

Partial Correctness Assertions

Triples $\{\mathtt{A}\}\mathtt{S}\{\mathtt{B}\}$, where $\mathtt{A}$ and $\mathtt{B}$ are assertions and $\mathtt{S}$ is a statement, are called *partial correctness assertions*. They are called "partial" because they allow statements to be undefined. Formally, $\{\mathtt{A}\}\mathtt{S}\{\mathtt{B}\}$ is *valid* iff for any state σ satisfying the assertion $\mathtt{A}$, if the statement $\mathtt{S}$ is defined in state σ and ends up in σ' then s' satisfies $\mathtt{B}$. In particular, the partial correctness assertion $\{\mathtt{true}\}$ `while(true) skip` $\{\mathtt{false}\}$ is valid because the while loop will never terminate.

We let $\sigma \models \mathtt{A}$ denote the fact that state σ satisfies $\mathtt{A}$. Then $\{\mathtt{A}\}\mathtt{S}\{\mathtt{B}\}$ is valid, written $\models \{\mathtt{A}\}\mathtt{S}\{\mathtt{B}\}$, iff

$$(\forall \sigma \in \Sigma)\ (\sigma \models \mathtt{A} \text{ and } [\![\mathtt{S}]\!]\sigma \text{ defined} \Rightarrow [\![\mathtt{S}]\!]\sigma \models \mathtt{B}).$$

To simplify writing, by convention we assume that "undefined" satisfies any assertion, so the above transforms into:

$$(\forall \sigma \in \Sigma)\ (\sigma \models \mathtt{A} \Rightarrow [\![\mathtt{S}]\!]\sigma \models \mathtt{B}).$$

We can now formally state the correctness of a statement as a partial correctness assertion, where the pre-condition gives the conditions under which the statement is executed and the post-condition gives the state properties at the end of the execution of the statement, in case that is reached. There are situations in which one may want to work with so-called *total correctness assertions*, often written `[A]S[B]`, which require the statement `S` to also be defined. We only consider partial correctness assertions in this lecture.

Hoare Logic

Hoare logic is defined as a set of inference rules for deriving valid correctness assertions. Like in the case of SOS, one can start with basic facts and then derive complex properties.

For the `skip` statement, the following is natural:

$$\frac{\cdot}{\{A\}\ \texttt{skip}\ \{A\}}$$

For an assignment $x = a$, a post-condition `B` holds if and only if `B` in which each occurrence of x is replaced by a holds as a pre-condition:

$$\frac{\cdot}{\{B[x \leftarrow a]\}\ \texttt{x = a}\ \{B\}}$$

We do not get into the technicalities underlying the formalization of assertions, but it is worth noticing that $B[x \leftarrow a]$ replaces only the *free occurrences* of x in B by the arithmetic expression a, and that bound variables in B *may need to be renamed* in order to avoid capturing names that occur free in a.

For sequential composition, one needs to "discover" an intermediate assertion $\mathtt{C}$:

$$\frac{\{A\}\ \mathtt{s_1}\ \{C\}, \quad \{C\}\ \mathtt{s_2}\ \{B\}}{\{A\}\ \mathtt{s_1;\ s_2}\ \{B\}}$$

In the case of conditionals, one has to treat the cases in which the condition is true and false, respectively:

$$\frac{\{A \wedge b\}\ \mathtt{s_1}\ \{B\}, \quad \{A \wedge \neg b\}\ \mathtt{s_2}\ \{B\}}{\{A\}\ \mathtt{if\ b\ then\ s_1\ else\ s_2}\ \{B\}}$$

Loops are the hardest to prove in practice, because one has to *discover an invariant assertion*, which sometimes is a highly non-trivial task. By analogy to mathematics, one can think of invariant discovery as "lemma discovery":

$$\frac{\{A \wedge b\}\ \mathtt{s}\ \{A\}}{\{A\}\ \mathtt{while(b)\ s}\ \{A \wedge \neg b\}}$$

A is called a *loop invariant*, because the body of the loop preserves it whenever it executes, that is, the condition holds. The invariant will then become false at the end of the loop.

Like in mathematics, where it is sometimes easier to prove a more general result then to prove directly the more particular one, we are often able to more easily prove more general partial correctness assertions. The following allows us to use a more general fact in order to derive a less general one:

$$\frac{\models A \Rightarrow A', \quad \{A'\}\ \mathtt{s}\ \{B'\}, \quad \models B \Rightarrow B'}{\{A\}\ s\ \{B\}}$$

We will stay informal with respect to the formal language for

assertions, and also implicitly with respect to what validity means for assertions. Intuitively, A is valid, written $\models A$, if and only if it holds in any state.

We write $\vdash \{A\}\ s\ \{B\}$ whenever $\{\mathtt{A}\}\ \mathtt{s}\ \{\mathtt{B}\}$ is *derivable* using the rules above. One can show the following important *soundness* result for *Hoare logic*:

Theorem. $\vdash \{A\}\ s\ \{B\}$ *implies* $\models \{A\}\ s\ \{B\}$.

The proof can be done by showing that each inference rule is sound.

Example

The following example shows the correctness of the program discussed at the beginning of this lecture, adding the first p natural numbers. The proof is quite detailed and, except the proofs of validity of state assertions, it explicitly shows all the derivation steps. We start by splitting the proof task into two subtasks:

$$\frac{PCA_1, \quad PCA_2}{\texttt{\{p>=0\} s=0; n=1; while(n<=p) (s=s+n; n=n+1) \{s==p*(p+1)/2\}}}$$

where PCA_1 is a partial correctness assertion derived as follows:

$$\frac{PCA_3, \quad PCA_4}{\texttt{\{p>=0\} s=0; n=1 \{p>=0} \wedge \texttt{s==0} \wedge \texttt{n==1\}}}$$

where PCA_3 is derived as follows:

$$\frac{\models \texttt{p>=0} \Rightarrow (\texttt{p>=0} \wedge \texttt{0==0}), \quad \frac{\cdot}{\texttt{\{p>=0} \wedge \texttt{0==0\} s=0 \{p>=0} \wedge \texttt{s==0\}}}}{\texttt{\{p>=0\} s=0 \{p>=0} \wedge \texttt{s==0\}}}$$

and PCA_4 as follows:

$$\frac{\models \texttt{(p>=0} \wedge \texttt{s==0)} \Rightarrow \texttt{(p>=0} \wedge \texttt{s==0} \wedge \texttt{1==1)}, \quad \left(\frac{\cdot}{\{\texttt{p>=0} \wedge \texttt{s==0} \wedge \texttt{1 == 1}\}\ \texttt{n=1}\ \{\texttt{p>=0} \wedge \texttt{s==0} \wedge \texttt{n==1}\}}\right)}{\{\texttt{p>=0} \wedge \texttt{s==0}\}\ \texttt{n=1}\ \{\texttt{p>=0} \wedge \texttt{s==0} \wedge \texttt{n==1}\}}$$

In order to show the partial correctness assertion PCA_2, that is, $\{\texttt{p>=0} \wedge \texttt{s==0} \wedge \texttt{n==1}\}$ `while(n<=p)(s=s+n;n=n+1)` $\{\texttt{s==p*(p+1)/2}\}$, we pick the invariant `A` in the rule associated to while loops to be `n<=p+1` $\wedge$ `s==n*(n-1)/2`. We first have to show that `A` is indeed an invariant, that is, that $\{\texttt{A} \wedge \texttt{n<=p}\}$ `s=s+n;n=n+1` $\{\texttt{A}\}$:

$$\frac{PCA_5,\ \{\texttt{n+1<=p+1} \wedge \texttt{s==(n+1)*((n+1)-1)/2}\}\ \texttt{n=n+1}\ \{\texttt{n<=p+1} \wedge \texttt{s==n*(n-1)/2}\}}{\{\texttt{n<=p+1} \wedge \texttt{s==n*(n-1)/2} \wedge \texttt{n<=p}\}\ \texttt{s=s+n;n=n+1}\ \{\texttt{n<=p+1} \wedge \texttt{s==n*(n-1)/2}\}}$$

where PCA_5 can be derived as follows:

$$\frac{\models \texttt{(n<=p+1} \wedge \texttt{s==n*(n-1)/2} \wedge \texttt{n<=p)} \Rightarrow \texttt{B}, \quad PCA_6}{\{\texttt{n<=p+1} \wedge \texttt{s==n*(n-1)/2} \wedge \texttt{n<=p}\}\ \texttt{s=s+n}\ \{\texttt{n+1<=p+1} \wedge \texttt{s==(n+1)*((n+1)-1)/2}\}}$$

where `B` is the assertion `n+1<=p+1` $\wedge$ `s+n==(n+1)*((n+1)-1)/2`,

and PCA_6 is the partial condition assertion:

$$\frac{\cdot}{\{\texttt{B}\}\ \texttt{s=s+n}\ \{\texttt{n+1<=p+1}\ \wedge\ \texttt{s==(n+1)*((n+1)-1)/2}\}}$$

Therefore, the assertion `A` above is indeed an invariant, so by the Hoare rule of the while loop we can derive

$$\frac{\{\texttt{A}\ \wedge\ \texttt{n<=p}\}\ \texttt{s=s+n;n=n+1}\ \{\texttt{A}\}}{\{\texttt{A}\}\ \texttt{while(n<=p)}\ \texttt{(s=s+n;n=n+1)}\ \{\texttt{A}\ \wedge\ \neg\texttt{(n<=p)}\}}$$

PCA_2 now can be derived by noting that

$$\models\ \texttt{(p>=0}\ \wedge\ \texttt{s==0}\ \wedge\ \texttt{n==1)}\ \Rightarrow\ \texttt{(n<=p+1}\ \wedge\ \texttt{s==n*(n-1)/2)}$$

and that

$$\models\ \texttt{(n<=p+1}\ \wedge\ \texttt{s==n*(n-1)/2}\ \wedge\ \neg\texttt{(n<=p))}\ \Rightarrow\ \texttt{s==p*(p+1)/2}$$

The hard part in proofs in Hoare logic is to find the appropriate invariants. In the example above, one would not be able to do the proof if one considered the invariant `A` to be just `s==n*(n-1)/2` instead of `n<=p+1 ∧ s==n*(n-1)/2`. The extra condition `n<=p+1`

may seem vacuous, but notice that it is very important to prove the partial correctness assertion for the while loop.

Exercise 1 *Prove the following partial correctness assertion using the Hoare logic inference rules:*

```
{x == n and n >= 0 and y == 1}
  while(x > 0) (
    y = x * y ;
    x = x - 1)
{y = n!}
```

where $n! = 1 * 2 * \cdots * n$ *is the factorial of* n*. As usual, do all the state assertion validity computations by hand and only show the applications of the Hoare inference rules.*

Exercise 2 *Prove the following partial correctness assertion using the Hoare logic inference rules:*

```
{x == m and y == n and n >= 1 and z == 1}
  while (y >= 1) (
    while (even?(y)) (
      x = x * x ;
      y = y / 2) ;
    z = z * x ;
    y = y - 1)
{z = m^n}
```

where `m^n` *is the power of* `m` *to the* `n` *and* `even?(y)` *tests whether* `y` *is an event number. As usual, do all the state assertion validity computations by hand and only show the applications of the Hoare inference rules.*